LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

Versionseuranta web-sovelluksen toteuttaminen

Ristimäki, Matti

2017 Laurea

LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

Versionseuranta web-sovelluksen toteuttaminen

Matti Ristimäki
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Toukokuu, 2017

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittelyn koulutusohjelma
Tradenomi (AMK)

Matti Ristimäki

Versionseuranta web-sovelluksen toteuttaminen

Vuosi 2017 Sivumäärä 35

Toiminnallinen opinnäytetyö tehtiin Suomen Pankille ja sen tavoitteena oli kehittää web-sovellus Wallstreet Suite (WSS) pankkijärjestelmän ylläpidon tarpeisiin. Sovelluksen tarkoiteuksena on automatisoida prosessia järjestelmän ympäristöjen asennustietojen kirjaamisessa, jotta tiedot ovat ajan tasalla sekä helposti vertailtavissa. Opinnäytetyön tavoitteena on analysoida kehitystyössä ilmenneet haasteet ja raportoida hankkeen vaiheet ja tekniset ratkaisut.

Tietoperustana opinnäytteessä on web-sovellusten kehitys ja kehityksessä keskeisessä roolissa olleet ASP.NET ja MVC-malli. Aineisto koostuu kehitysopaskirjoista, sähköisistä artikkeleista ja hankkeen aikana syntyneestä materiaalista. Kehitystä ja infrastruktuurin luonnissa oli mukana järjestelmäasiantuntijoita, joiden kanssa työskentely mahdollisti useimpien teknisten haasteiden ratkaisun.

Tuotoksena syntyi vakaa alusta WSS-pankkijärjestelmän tietojen seurantaan, jonka päälle voi hyvin jatkokehittää uusia ominaisuuksia ylläpidon tarpeiden mukaan.

Asiasanat: web-sovellus, ASP.NET, MVC-malli, web-kehitys

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor's Thesis

Matti Ristimäki

Developing version control web application

Year 2017 Pages 35

This thesis was done for Bank of Finland. The goal was to develop a web-application to support WSS bank system administration team. The web application came in to automate an existing manual process of keeping up with system's environments installation packages. With the application environment data would be easy to maintain and compare to each other. This thesis reports the steps of making the web-application and what challenges were overcome.

The frame of reference for the thesis contains the methods of making a web application. The material was gathered from web development guides, during the process of making the application and from digital articles of the subject. Working with system specialists helped gain better understanding of building the infrastructure and solving technical challenges.

As a result, the web application is a solid foundation for keeping and with the systems environments and for the WSS administration and support teams to build upon.

Keywords: web application, ASP.NET, MVC pattern, web development

Sisällys

1 Johdanto

Tämän toiminnallisen opinnäytetyön tavoitteena oli suunnitella ja kehittää Suomen Pankin Wallstreet Suite (WSS) pankkijärjestelmään automaattinen versionseurantasovellus. Opinnäytetyö kuvaa web-sovelluksen kehitysprosessia ja käsittelee siinä eteen tulleita haasteita sekä oivalluksia. Sovelluksen tarkoituksena on tukea WSS-pankkijärjestelmän ylläpitoa, pitämällä ympäristön tiedot ajan tasalla ja helposti saatavilla. Sovelluksen kärki tavoitteena oli myös helpottaa ympäristöjen keskinäistä vertailua.

Kehitysprosessin edetessä päädyttiin sovellus toteuttamaan web-pohjaisen sovelluksena. Web-sovellukset tarjoavat huomattavia etuja työpöytäsovelluksiin nähden niiden joustavuuden sekä helpon jakelun ansiosta. Päivitykset sovelluksiin tapahtuvat palvelin puolella, joten pieniäkin päivityksiä pystytään tarjoamaan käyttäjille nopeasti sekä ilman erillisiä asiakkaalle näkyviä päivityksiä tai asennuksia. Ne ovat hyvä ratkaisu yrityksen sisäisten haasteiden ja tarpeiden täyttämiseen kohtuullisella budjetilla ja ajallisella sijoituksella, varsinkin jos yrityksen infrastruktuuri tukee jo valmiiksi web-sovellusten kehitystä ja ylläpitoa.

Prosessina web-sovelluksen kehitys on monialainen ja se vaatii osaamista useasta aiheesta, kuten tietokannoista, verkkoliikenteestä ja käyttöliittymäsuunnittelusta verkkosivujen perustaitojen kuten HTML, JavaScriptin ja CSS lisäksi. Kuitenkaan kaikkia osa-alueita ei voi osata niiltä syvän erikoistumisen takia, joten sovelluskehityksessä vaaditaan tiimityötä useiden kehittäjien ja asiantuntijoiden välillä.

Tässä hankkeessa web-sovelluksen kehityksessä käytettiin paljon Microsoftin tuotteitta ja sovellus perustuukin Microsoftin ASP.NET MVC kehitysmalliin. Microsoftin tuotteiden yhteensopivuus nopeutti kehitysprosessia ja varsinkin toimeksiantajan laajaosainen tuntemus tuotteista vahvisti sovelluksen toimivuuden luotettavuutta, sillä suurin osa implementoiduista ominaisuuksia olemassa olviin järjestelmiin olivat jo ennestään tuttuja.

2 Työn tausta ja tavoitteet

Tässä kappaleessa kerrotaan hankkeen opinnäytetyön toimeksiantajasta, Suomen Pankista, ja sen keskeisestä toiminnasta ja mihin sovelluksen kehittäminen sijoittuu organisaatiossa. Lisäksi määritellään opinnäytetyön ja hankkeen tavoitteet.

2.1 Toimeksiantaja

Työ tehdään Suomen Pankille. Suomen Pankki on Suomen keskuspankki ja toimii osana eurojärjestelmää. Suomen Pankki osallistuu euroalueen rahapolitiikan valmisteluun, rahanhuol-

toon ja rahoitusmarkkinoiden vakauden valvontaan. Suomen Pankin toimintaa valvoo eduskunnan pankkivaltuusto. Finanssivalvonta toimii Suomen Pankin yhteydessä (Suomen Pankki 2015.)

Suomen Pankilla on oma IT-yksikkö, jonka tehtävänä on tarjota tietotekniikkapalveluita organisaatiolle ja varmistaa, että informaatioteknologian hyödyntämisessä noudatetaan kansallisen keskuspankin ja eurojärjestelmän toiminnan edellyttämiä laatu- ja turvallisuusvaatimuksia (Suomen Pankki 2015). Yksikkö koostuu kolmesta toimistosta: infra-, järjestelmä- ja palvelunhallintatoimistoista. Tämä opinnäytetyö tehdään IT-yksikön infratoimistoille, joka on vastuussa Wallstreet Suite -järjestelmän ylläpidosta Suomen Pankissa.

2.2 Tavoite

Opinnäytetyön tavoitteena on automatisoida prosessia, jota käytetään Wallstreet Suite (WSS) järjestelmään asennettujen pakettien versioiden seurantaan. Nämä paketit ovat päivityksiä ympäristön ominaisuuksiin sekä turvallisuuteen. Järjestelmän toimivuuden takia on tärkeää, että tiedot ovat ajan tasalla. WSS-järjestelmä koostuu monesta eri ympäristöstä, jotka jakautuvat edelleen asiakas-palvelin-arkkitehtuuriin.

Nykyinen tapa ympäristöjen tietojen päivitykseen ja ylläpitoon on Excel-taulukko, jota päivitetään käsin Pankin intranetiin. Järjestelmän kanssa työskentelee liiketalouden puolella useampi ihminen, mutta IT-osastolla ylläpidosta vastaa vain muutama. Koska resursseja ei ole paljon, on Excel-taulukon ylläpitäminen työlästä ja sen päivittäminen saattaa kiireessä unohtua. Tavoitteena on toteuttaa nopeasti yksinkertainen ja helppokäyttöinen ratkaisu version seurantaan.

Opinnäytetyö kuvaa web-sovelluksen kehitysprosessia ja siinä eteen tulleita haasteita. Tarkoituksena on kuvata web-sovellusta arkkitehtuuriselta puolelta kuin myös nopean kehityksen kannalta. Opinnäytetyössä tutkitaan ASP.NET kehitystä, sekä web-sovelluksissa suosittua MVC-mallia. Suunniteltavan ja kehitettävän sovelluksen ei oleteta kilpailevan valmiiden versionseuranta sovellusten toimintojen kanssa, vaan tarkoituksena on tehdä nopea ja yksinkertainen ratkaisu Suomen Pankin tarpeisiin, jota voidaan jatkossa kehittää tarpeen mukaan.

3 Käsitteet

HTTP

HyperText Transfer Protocol (HTTP) on yksi World Wide Webin peruskivistä. Se perustuu pyyntöihin (request) ja vastauksiin (response). Asiakasohjelma (client) lähettää palvelimelle (server) HTTP pyynnön johon palvelin vastaa HTTP viestillä, joka sisältää sisältöä. HTTP on tilaton (stateless) protokolla. (Shklar & Rosen 2009, 33.)

HTML

Hyper Text Markup Language (HTML) on yleinen nettisivujen standardi, jota on käytetty WWW:n alusta asti. HTML kuvaa sivuston rakennetta elementtien avulla, jotka muodostuvat merkinnöistä eli tageista. (W3C.)

CSS

Cascading Style Sheet (CSS) kuvaa nettisivujen tyyliä. Se helpottaa pitämään useiden sivujen tyylit yhtenäisinä linkittämällä sama tyylitiedosto kaikkiin sivuihin. HTML ja CSS luovat yhdessä pohjan verkkopohjaiselle käyttöliittymärakenteelle ja -ulkoasulle. (W3C.)

JavaScript

JavaScript on monialustainen ohjelmointikieli, jota käytetään front-end sekä back-end puolen ohjelmointiin. Verkkosivujen kehittämisessä sitä käytetään elementtien muokkaamiseen ja hallinnointiin (Mozilla Developer Network). Sen suurin vahvuus on sivujen sisällön päivitys ilman että sivua tarvitsee ladata uudestaan. Vaikka JavaScript suunniteltiin aluksi HTML lomakkeiden hallitsemiseen, on siitä tullut yksi suosituimmista ohjelmointikielistä sen monipuolisuuden ansiosta.

Framework

Framework on alusta, joka tarjoaa ohjelmoijille perustan, jolla kehittää ohjelmia. Sen tarkoituksena nopeuttaa ohjelmistojen kehittämistä tarjoamalla valmiita metodeja ja luokkia, kuin myös laitteisto ja käyttöjärjestelmä kohtaisia elementtejä. Frameworkit voivat olla isoja kokonaisuuksia kuten Microsoftin .NET Framework tai koostua pienemmistä määristä hyödyllisiä metodeita kuten verkkosivujen ulkoasuun ja käyttöliittymien kehitykseen suunnattu Skeleton-framework. (Techterms 2013.)

.NET Framework

.NET Framework on Microsoftin kehittämä ohjelmointi alusta, jonka tarkoituksena on suoraviivaistaa ja helpottaa Windows ohjelmien kehitystä. Ennen .NET:ä Windows ohjelmia kehitettiin C tai C++ ohjelmointikielillä. C:n haastavuuteen luetaan manuaalinen muistin käytön ohjelmointi ja strukturaalinen rakenne, jotka ovat väistyneet olio-ohjelmoinnin tieltä. .NET Framework koostuu CLR (Common language runtime) moottorista, jolla suoritetaan ohjelmia, sekä luokkakirjastosta, joka sisältää valmiita luokkia ja koodia, mitä kehittäjät voivat käyttää ohjelmien kehityksessä. (Microsoft.)

Razor

Razor on ASP.NET MVC:n view-moottori. Sen avulla HTML merkinnän sekaan voidaan lisätä C# tai Visual Basic koodia, jota .NET Framework tukee. Razor tiedostomuoto on .cshtml, joka erottaa ne staattisista .html tiedostoista. Kun C#-koodia haluaa kirjoittaa HTML-sivulle, onnistuu se @-merkin jälkeen, jolloin Razon tunnistaa, että kyseessä on koodia, eikä HTML:ää. (Hexter, Palermo, Skinner, Bogard, & Hinze 2012, 6) Razor otettiin oletuksena ASP.NET MVC mukaan versiossa 3 vuonna 2011 (Microsoft).

LINQ

Language Integrated Query (LINQ) on kieli, jolla pystytään hakemaan tietoa monesta eri tietolähteestä, kuten luokista. Kielenä LINQ on saman tyylinen kuin SQL-kieli. (Freeman 2013, 86). Niiden syntaksit ovat molemmat helposti luettavia. LINQ julkaistiin osaa .NET 3.5 versiota. (Hexter, Palermo, Skinner, Bogard, & Hinze 2012, 34.)

SSIS

SQL Server Integration Service (SSIS) on Microsoftin SQL Server tuoteperheen ohjelma, jolla automatisoidaan dataan liittyviä tehtäviä. Sitä käytetään datan keräämiseen eri tietolähteistä, datan lisäämiseen tietokantaan ja tietokantatehtävien suorittamiseen (Microsoft). SSIS-paketin luominen ei vaadi koodausta vaan sen logiikka muodostetaan yhdistämällä valmiita tehtäviä käyttöliittymässä.

4 Web-sovelluksen kehittäminen

Web-sovellukset perustuvat client-server arkkitehtuuriin. Selain toimii asiakkaana (client), joka lähettää pyyntöjä palvelimelle (server), jossa sisältö muodostetaan ja lähetetään vastauksena selaimelle. Web-sovelluksen suurin ero perinteiseen työpöytäsovellukseen on selaimen käyttö ohjelman alustana. (Shklar & Rosen 2009, 201.)

Web-sovellusten kehittäminen vaatii laajaa osaamista monelta osa-alueelta. Kehittäjän tulee tuntea ohjelmointikieliä kuten Javascriptiä, HTML-kieltä, sekä protokollia, kuten HTTP:tä. Lisäksi tuntemusta tulee olla relaatiotietokannoista, graafisesta suunnitellusta ja mediasta. (Shklar & Rosen 2009, 201) Koko ketjun osaavia kehittäjiä kutsutaan ”full stack”-kehittäjiksi.

4.1 Web-sovellusten suosio

Web-sovellusten kehittäminen on yleistynyt niiden joustavuuden ja helppokäyttöisyyden ansiosta. Ne tarjoavat suuria etuja työpöytäsovelluksiin nähden: sovelluksia ei tarvitse asentaa ja melkein jokainen moderni laite sisältää selaimien, jolla käyttää sovellusta. Web-sovellukset pystyvät useassa tapauksessa myös samaan kuin työpöytä vastineensa, esimerkiksi Google Sheets verrattuna Microsoft Exceliin. (Bychkov 2013.)

Selainpohjaisen sovelluksen jakaminen käyttäjille on vaivatonta. Käyttäjän ei tarvitse joka kerta asentaa ohjelmaa uudestaan vaihtaessa työasemaa tai päivittää sitä, kun ohjelmistotoimittajalla on uusi versio saatavilla (Bychkov 2013). Siinä missä työpöytäsovellukset voivat vaatia erillisiä ohjelmistolisäosia tai laitteistokohtaisia ominaisuuksia toimiakseen, vaativat websovellukset useimmissa tapauksissa vain modernin verkkoselaimen. Sovelluksen logiikka ja tarvittavat komponentit ovat web-sovelluksen taustalla olevilla palvelimella (Kovalick 2014). Näin sovelluksista voidaan rakentaa kevyitä, jolloin laitteistorajoitteiset käyttäjät pääsevät suorittamaan vaativampia operaatiota. Niistä voidaan myös tehdä käyttäjän laitteistoa hyödyntäviä sovelluksia, jolloin kuormitusta ei tule yhtä paljon sovelluspalvelimille. (Bychkov 2013.)

Web-sovellukset sopivat moneen eri käyttötarkoitukseen, mutta ne ovat parhaimmillaan, kun niitä ei käytetä kokoaikaisesti monimutkaisten tehtävien suorittamiseen, kuten suunnittelija voisi käyttää esimerkiksi Photoshopia työn päätyökaluna. Kommunikaatiotyökaluja kuten Skypeä ja Slackia valmistetaan vielä työpöytäsovelluksinakin, koska niiden pääsääntöiset käyttäjät tarvitsevat nopean pääsyn sovellukseen, eikä selaimen välilehdet tai muut ominaisuudet häiritse käyttöä. (Collin 2015.) Web-sovellukset toimivatkin hyvin käyttötapauksissa, jossa sovellukseen kohdistuva huomio ei ole jatkuvaa.

4.2 Nopeasti valmista Bootstrapilla

Web-sovelluskehityksessä verkkosivu toimii käyttöliittymänä. Käyttöliittymien kehitykseen on tarjolla useita frameworkeja, joista suosituimpia on Twitter Inc:n kehittämä Bootstrap. Se sisältää valmiita CSS-tyylitiedostoja ja JavaScript-toimintoja verkkosivujen kehitykseen. Se mahdollistaa skaalautuvien eli responsiivisten verkkosivujen luonnin ruudukko-ominaisuudella, jossa HTML-elementit järjestetään näytön koon mukaan päällekkäin tai vierekkäin. (Spurlock 2013, 3) Bootstrap on suosittu web-kehittäjien keskuudessa, koska sen avulla on nopea lähteä

työstämään uusia projekteja. Kehittäjä saa nopean käyttöliittymäratkaisun tiedon esittämiselle ja kestävän alustan, jonka päälle rakentaa (Spurlock 2013, 1).

Bootstapin suuri suosio tuo mukanaan suuren tiedonmäärän, josta löytyy ratkaisu melkein kaikkiin ongelmiin. Bootstrapin kehityskin tapahtuu avoimesti GitHubissa ja se jatkuu koko ajan (Spurlock 2013, 99). Tällä hetkellä Bootstap on versiossa 3.3.7, mutta uusi 4. versio on kehitteillä ja sen keskeneräinen testiversio on avoin kaikille (Bootstarp Blog 2017).

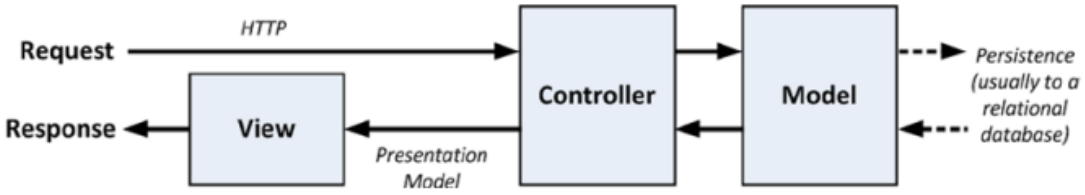
Bootstrap on iso kokonaisuus erilaisia komponentteja, joiden kaikkien lisääminen projektiin voi tehdä siitä raskaan. Vaikka monet pitävät tätä heikkoutena, Stevensin mukaan Bootstrapia pitää ajatella avoimena buffettina, josta kehittäjän pitää valita tarvittavat komponentit projektiaan varten. Tähän ratkaisuna Bootstrap tarjoaa sivuston, josta kehittäjä voi valita projektissaan tarvittavat komponentit ennen paketin lataamista. (Stevens 2014.)

Toisena heikkoutena pidetään Bootstrapillä kehitettyjen sivustojen saman tyylisyyttä ja näköisyyttä. Tähän Steven osoittaa ratkaisuna tarkemman komponenttien valinnan, jonka päälle on helppo ylikirjoittaa uusia tyylejä. Valmiiden pohjien käyttäminen kuitenkin auttaa sovelluksen alkuvaiheessa, jos projektilla ei ole graafista suunnittelijaa tai projekti tulee saada nopeasti valmiiksi pienempään käyttöön. (Stevens 2014.)

4.3 MVC-malli

Model View Controller (MVC) on ohjelmointimalli, jota käytetään pääasiassa web-sovellusten kehittämiseen. Malli kehitettiin jo 1970-luvulla Smalltalk projektissa Xerox PARC:ssa. Se oli kuitenkin termeiltään sijoittunut pitkälti Smalltalkin omiin konsepteihin, jonka laajempi konsepti otettiin käyttöön pitkälti web-sovellusten kehitykseen (Freeman 2013, 51.) Sen toi laajemmalle yhteisölle web-sovellusten kehitykseen Ruby on Rails vuonna 2003, ja Microsoft otti sen käyttöön ASP.NET kehitysmallissaan vuonna 2007 (Hexter, Palermo, Skinner, Bogard, & Hinze 2012, 6.)

MVC-mallissa verkkosivuston tai web-sovelluksen palvelinlogiikka ja käyttöliittymä pakotetaan erilleen toisistaan. Model, View ja Controller ovat erillisiä kokonaisuuksia, jotka käsittelevät vain niille tarkoitettuja tehtäviä: Model käsittelee dataa, View käsittelee tiedonnäyttämisen logiikkaa ja Controller käsittelee käyttäjän syötteen (Kuvio 1) ja siihen liittyvän logiikan. Web-sovelluksessa tämä helpottaa sovelluksen kehitystä, kun HTML-sivusto on eriytetty. (Freeman 2013, 54.)



Kuvio 1: MVC-mallin logiikka (Freeman 2013, 53.)

Model määrittää sovelluksen toimialueen, jotka kuvastetaan luokkina. Modelit ovat yksinkertaisuudessaan olio-ohjelmoinnista tuttuja luokkia. Ne kuvastavat tausta olevaa dataa, sekä dataan tehtäviä operaatioita. Esimerkiksi pankkisovelluksessa ne voisivat kuvata tilejä ja niillä tehtäviä tapahtumia kuten nostoja ja panoja. Modelit myös varmistavat datan eheyden, kun sen kanssa työskennellään. (Freeman 2013, 52.)

Viewit kuvaavat modelien dataa visuaalisesti ja sisältävät tarpeen mukaan logiikkaa siihen. Viewit eivät kuitenkaan vastaan datan muokkaamisesta – se tapahtuu modelissa. Web-sovelluksissa viewit koostuvat yleensä HTML merkinnästä. (Hexter ym. 2012, 7.) Logiikan kirjoittamiseen voidaan käyttää esimerkiksi C#:a kuten Microsoftin ASP.NET MVC:ssä tai jotain muuta ohjelmointikieltä kuten PHP:ta Ruby on Railsissa.

Controller toimii sovelluksen ytimenä. Se liittää Modelit ja Viewit toisiinsa. Controller ottaa vastaan web-sovellukseen tehtävät http-kutsut ja käsittelee ne sisälletyn logiikan mukaan. Yleensä http-kutsut johtavat viewin muodostamiseen tai siirtymiseen toiselle sivulle. Controller myös suorittaa modelien operaatiot, kuten pankkisovelluksessa tililtä noston. (Freeman 2013, 52.)

MVC-malli on tarkka kansiorakenteesta ja tiedostojen nimeämisestä. Controllerien nimien pitää alkaa sen tarkoituksella ja loppua sanaan Controller, esimerkiksi HomeController. Kansioiden osalta Controllers ja Views kansiot ovat pakollisia. Views kansion alikansioiden tulee mukailla niitä ohjaavaa Controllerin nimen alkua, esimerkiksi polku voisi olla Views/Home/. (Hexter ym. 2012, 18.)

4.4 ASP.NET web-sovelluskehityksessä

ASP.NET on Microsoftin kehittämä avoimen lähdekoodin verkkokehitysmalli, joka hyödyntää Windows alustoille rajoitettua ja sidottua .NET Frameworkia. ASP.NET mahdollistaa nettisivujen kehittämisen ohjelmointikielillä, jotka ovat yhteensopivia Common language runtime (CLR) moottorin kanssa. Näitä ovat esimerkiksi Microsoftin Visual Basic ja C# ohjelmointikie-

let. ASP.NET tarjoaa kolme toisistaan eroavaa alustaa web-sovellusten ja verkkosivujen kehitykseen: ASP.NET Web Forms, ASP.NET Web Pages ja ASP.NET MVC. Jokaisella alustalla on oma kohderyhmänsä.

ASP.NET Web Forms oli ensimmäinen verkkokehitysmalli ASP.NET alustalle. Sen avulla kehittäjät pystyivät rakentamaan helposti käyttöliittymiä, sen ”raahaa ja pudota” tyylisen editorin avulla. Se pohjautuu suuresti Windows Formsiin, joka on tarkoitettu työpöytäsovellusten kehitykseen. Web Forms kuitenkin oli suuri haaste kehittäjille, koska se teki verkkosivuista tilallisia (stateful) kun verkkosivut ovat yleensä ovat tilattomia (stateless). (Hexter ym. 2012, 5)

ASP.NET Web Pages julkaistiin osana ASP.NET MVC julkaisua, ja on suunnattu kevyiden verkkosivujen kehittämiseen. Web Pages tarjoaa kuitenkin samoja ASP.NET ominaisuuksia, mutta ei ole yhtä monimutkainen kehittää kuin MVC malli (Hexter ym. 2012, 5). ASP.NET MVC tarjoaa kolmesta vaihtoehdosta ketterimmän sekä nykyaikaisemmin kehitysmallin perustuen MVC-malliin. Toisin kuin kaksi muuta alustaa, ASP.NET MVC on avoimen lähdekoodin alusta. (Freeman 2013, 12.)

Microsoftin ASP.NET muodostaa noin 15,1 % palvelinpuolen ohjelmointi kielistä (W3Techs 2017). Tämä tulee esille W3Techsin tekemästä tilastosta, joka analysoi 10 miljoonaa verkkosivua ja niiden käyttämää ohjelmointi kieltä, jonka pystyy havaitsemaan. Verkkosivujen tiedot tulevat Amazon.comin ylläpitämästä Alexa data-palvelusta. (W3Techs 2017.) Suurin osa verkkosivustoista käyttää PHP-ohjelmointikieltä, jonka osuus on noin 82,6 %.

5 Tutkimusmenetelmät

Tässä kappaleessa käydään läpi keskeiset tutkimusmenetelmät, sekä tämän opinnäytetyön keskeinen malli, eli toiminnallinen opinnäytetyö. Esille nostetaan myös opinnäytetyön aineiston hankinta.

5.1 Kvantitatiivinen tutkimusmenetelmä

Kvantitatiivisessa, eli määrällisessä, tutkimusmenetelmässä keskitytään tutkimaan objektiivista tietoa tutkimus kohteesta. Tutkimuksessa kiinnostaa kohteen määrällinen ja numeerinen tieto, koska niiden perusteella oletetaan pystyvän selittämään monia ilmiöitä. Lopullinen tuotettu aineisto saatetaan tilastollisesti käsiteltävään muotoon (Hirsijärvi, Remes & Sajavaara 2004, 131).

Määrällisessä tutkimusmenetelmässä perehdytään aiempaan teoriaan tutkittavasta aiheesta. Teorian muodostamista pidetäänkin korvaamattomana tutkimukselle, koska se nopeuttaa pe-

rusperiaatteiden esittämistä tutkimuksessa. Teorian tueksi määritetään myös keskeiset käsitteet. Teorian pohjalta tutkija voi myös esittää perusteltuja hypoteeseja tuloksista joita odotetaan tai uskotaan saavan. (Hirsijärvi, Remes & Sajavaara 2004.)

5.2 Kvalitatiivinen tutkimusmenetelmä

Kvalitatiivisessa, eli laadullisessa tutkimusmenetelmässä pyritään kuvaamaan todellista elämää ja yleensä kohdetta kuvataankin kokonaisvaltaisesti (Hirsijärvi, Remes & Sajavaara 2004, 155). Tutkijaa kiinnostaa enemmän aineiston laadulliset määritelmät, joita on hankala mitata määrällisesti, kuten tunteita tai merkitystä eri tilanteissa.

Laadullisessa tutkimusmenetelmässä tärkeimpinä tiedon keruu välineinä pidetäänkin ihmisiä, sekä todellisen elämän tilanteita. Tutkija luottaa tuottamaansa aineiston yksilöllisyyteen, joita saavat haasteluista tai kyselyistä. Haastateltavat ihmiset valitaankin tarkoituksen mukaisesti. (Hirsijärvi, Remes & Sajavaara 2004, 155.)

5.3 Toiminnallinen opinnäytetyö

Opinnäytetyössä esitetään opiskelijan toimialan osaamista ja kehitystä. Toiminallisessa opinnäytetyössä toteutetaan yleensä hanke tai ollaan mukana hankkeen kehityksessä, joka tehdään toimeksiantajalle. Toimeksiantajana toimii yleensä yritys. Työ kostuu kehityshankkeen toteutuksesta, sekä opinnäytetyön raportoinnista. Toiminnallisen opinnäytteen tuloksena syntyy tuotos, joka parantaa edellistä tai on kokonaan uusi. (Salonen 2013, 25.)

Tiedonhankintamenetelmät mukailevat toiminnallisessa opinnäytetyössä paljon tutkimusmenetelmiä, vaikka niiden käyttö on joustavampaa (Salonen 2013, 23). Tässä opinnäytetyössä ainestoa on hankittu paljolti kehityshankeen havainnoista ja lähdekirjallisuudesta. Myös kehitysvaiheessa olleiden asiantuntijoiden suulliset neuvot ovat avustaneet uusien havaintojen ja ratkaisuiden löytämisessä.

5.4 Validiteetti ja reliabiliteetti

Validiteetti on tutkimuksen arviontiin liittyvä käsite, joka määrittää tutkimuksen pätevyyttä. Siinä mitataan, onko tutkimusmenetelmällä kykyä tuottaa oikeaa tietoa tutkimuksen kohteesta. Sen avulla päästään varmuuteen, että tutkija ei käsittele tutkimustuloksia oman ajattelumallinsa mukaisesti vaan ottaa huomioon esimerkiksi kvalitatiivisessa tutkimuksessa esiin nousevat useat tulkinnat. (Hirsijärvi, Remes & Sajavaara 2004, 217.)

Reliabiliteetilla ilmaistaan mittaustulosten toistettavuutta. Kvalitatiivisten tutkimuksien tuloksia voidaan pitää reliaabeleina, jos esimerkiksi kaksi tutkijaa päätyy samaan tulokseen samaa henkilöä tutkittaessa. Kvantitatiivisten tutkimusten luotettavuutta voidaan mitata erilaisilla tilastollisilla menettely tavoilla. (Hirsijärvi, Remes & Sajavaara 2004, 216.)

6 Määrittely

Tässä kappaleessa kerrotaan tarkempi hankekuvaus ja siihen liittyvät palvelin ja kehitysympäristö määrittelyt. Hankkeen aikataulu ei ollut tarkkaan määritelty, mutta sovelluksen piti olla nopeasti valmis. Sen suunnittelu ja toteutus tapahtuivat muiden operatiivisten töiden ohessa Suomen Pankin Infra-toimistossa.

Toteutukseen ja suunnitteluun osallistui järjestelmän vastuuhenkilöitä, joiden tehtävänä on järjestelmän ylläpito. Koska järjestelmä tulisi paljolti heidän työn tuekseen, oli heidän mukana olo hankkeen elinkaarenaikana hyödyllistä lopullisen käytön kannalta. Myös WSS-loppukäyttäjien suullinen palaute oli arvokasta sovelluksen käyttökokemuksen hahmottamisessa.

6.1 WSS-järjestelmän määrittely

Wallstreet Suite (WSS) järjestelmä on keskuspankeille suunnattu järjestelmä muun muassa varainhallintaan ja vakuuksien hallintaan (ION Group). Sen toteuttaja on Wall Street Systems Delaware, Inc, joka perustettiin vuonna 1986, mutta on toiminut ION Tradingin tytäryhtiönä vuodesta 2011 (Bloomberg).

WSS-järjestelmässä on monta ympäristöä, jotka voidaan jakaa kategorioihin: tuotanto, tuotannonkaltainen, testi ja kehitys. Nämä ympäristöt jakautuvat edelleen asiakas ja palvelin tyyppeihin. Ympäristöihin asennettuja ominaisuuksia ja päivityksiä kutsutaan paketeiksi, joiden seurantaan hankkeessa toteutettava sovellus tulee. Näiden pakettien asennus ei kuitenkaan kuulu opinnäytetyön laajuuteen, eikä asennuspakettien tekninen ymmärtäminen vaikuta hankkeen toteutukseen.

Ympäristöihin asennettuja paketteja pääsee tutkimaan WSS-järjestelmän omasta komentotulukista, eli shellista, joka tarjoaa useita komentoja ympäristön kanssa työskentelyyn. Sillä voidaan esimerkiksi listata asennettujen pakettien tiedot taulukkona, joka sisältää pakettien tiedot ja se on ympäristöstä riippumatta samassa muodossa. Lähtökohtana version valvonnan luomiselle on tämän tiedon esittäminen käyttäjäystävällisellä ja ympäristöjen vertailun kannalta helpolla tavalla.

6.2 Ympäristökuvaus

Suomen Pankissa työskennellään Microsoftin ympäristössä, joten hankkeen toteutustavan pitää olla yhteensopiva Microsoftin tuotteiden kanssa. Tarkoituksena on käyttää olemassa olevaa kehityspohjaa, eikä ottaa käyttöön Microsoftin ulkopuolisia ratkaisuja tähän hankkeeseen. Niiden käyttöönotto vaatisi laajaa selvitystyötä ja monimutkaistaisi ympäristön ylläpitoa. Käyttöjärjestelminä hankkeessa toimivat Windows Server 2012 ja Windows Server 2012 R2 ja loppukäyttäjien työasema käyttöjärjestelminä Windows 7, sekä projektin aikana käyttöön otettu Windows 10.

Kehitystyössä käytössä olevat työkalut ovat Microsoftin Visual Studio Professional 2013 ja SQL Server Management Studio 2012. Visual Studiossa käytetään myös SQL Server Data Tools 2012 lisäosaa, jolla mahdollistetaan Integration Services pakettien kehittäminen. Tietokantapalvelimet käyttävät Microsoftin SQL Server 2012:sta ja verkkosivujen julkaisujärjestelmänä käytössä on Microsoftin Internet Information Services eli IIS.

Vaikka Visual Studio tukee montaa eri ohjelmointikieltä. Web-sovellusten ja verkkosivujen työstämiseen käytetään Microsoftin ASP.NET verkkokehitys mallia, joka hyödyntää C#-koodia ja .NET Frameworkia. Tämä vaati kehittäjältä paljon uuden opettelemista, kun lähtökohtina olivat avoimen lähdekoodin Java ja PHP ohjelmointikielet. Koska ohjelmointikielet kuitenkin vastaavat paljon toisiaan on edellisen kokemuksen päälle helpompi rakentaa.

7 Toteutus

Hankkeen toteutusvaihe alkoi jo ennen kuin suurinta osaa määrityksistä saatiin kasaan. Kehittäjän on tärkeä ammentaa jo oppinutta tietoa työskentely ympäristöstä, jotta suunnittelu vaihe olisi sujuvampaa ja useammat haasteet osattaisiin ottaa jossain määrin huomioon. Toteutus vaiheen raportoinnissa käytetään hyväksi lähdekoodin esimerkkejä, sekä havainnollistavia kuvia niin suunnittelu vaiheesta kuin toteutuksen eri aloilta. Tärkeää on muistaa, että toteutuksen raportointi ei opeta sovelluskehitystä vaan keskitytään kuvaamaan sen eri vaiheita sekä haasteita.

Kehitys tapahtui Suomen Pankin kehitysympäristössä, joka on erillään tuotannon ja testin ympäristöistä. Kehitysympäristö mahdollistaa vapaamman testailun ja kokeilu. Huomioon on otettava se, että kehitettävän sovelluksen tulee olla yhteensopiva tuotannon ympäristön kanssa, joten erovaisuudet kehitys- ja tuotantoympäristöjen välillä tulee ottaa huomioon. Esimerkiksi kehitysympäristöön oli asennettu uudempi versio SQL Serveristä sekä .NET Frameworkista.

7.1 Sovelluksen suunnittelu

Suunnitteluvaihe käynnistyi sovelluksen perusideasta ja tarpeesta, joka on WSS-järjestelmään asennettujen pakettien helppo vertailu sekä tiedon pitäminen ajan tasalla. Lisäksi tehtiin muutamia rajauksia sovelluksen tarpeista ennen varsinaista käyttöliittymän suunnittelua. Sovellus tulee työasema käyttöön, joten esimerkiksi sen skaalautuminen mobiiliin ei ole tarpeellista. Todettiin myös, että vanhempien WSS-järjestelmän asennuspakettien tietoja ei tarvitse säilyttää, koska se tieto ei ole WSS-tukiryhmälle tarpeellista.

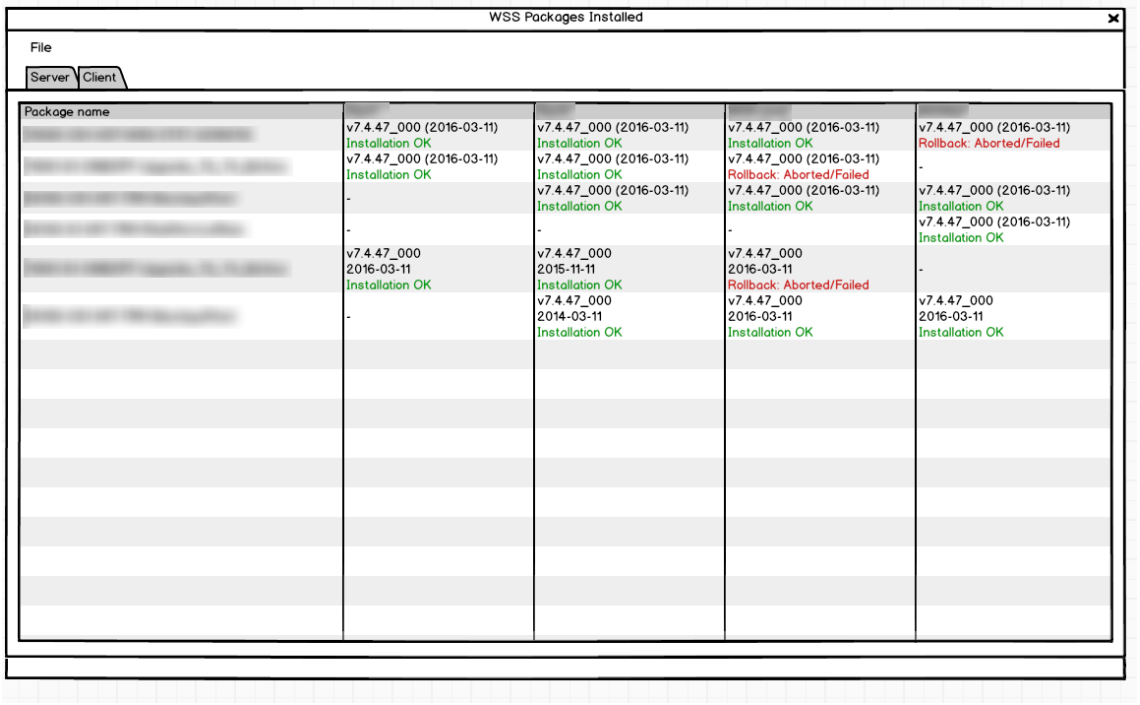
Käyttöliittymän idea hahmoteltiin aluksi paperille yhteistyössä ylläpidon kanssa. Tärkeimpään ominaisuuteen eli pakettien vertailuun keskenään keskityttiin suunnittelu vaiheessa eniten. Vertailutaulu määriteltiin seuraavasti: kaikki ympäristöön asennettujen pakettien nimet sijoitetaan oikealle ja vasemmalle muodostuu kaikkien ympäristöjen tiedot pakettien nimien perusteella. Jos ympäristöstä ei löydy oikean sarakkeen pakettia, jätetään se tyhjäksi. Tästä muodostui seuraava hahmotelma (Taulukko 1.)

Nimi	Ympäristö 1	Ympäristö 2	Ympäristö 3	Ympäristö 4
Paketti 1	versionumero päivämäärä asennustila	versionumero päivämäärä asennustila	versionumero päivämäärä asennustila	
Paketti 2		versionumero päivämäärä asennustila		versionumero päivämäärä asennustila

Taulukko 1: Hahmotelma vertailutaulukosta

Aluksi ideana oli tehdä C#-ohjelmointikielellä Windows työpöytäsovellus. Työpöytäsovellus tulisi jaella WSS-tukiryhmälle, sekä yksittäisille järjestelmän ylläpitäjille. Sen toteutus sopisi hyvin Microsoft ympäristöön koska se hyödyntäisi .NET frameworkia ja saisi käyttöönsä tehokkaita komponentteja.

Alkuperäisen piirustuksen ja määritelmien perusteella tehtiin Balsamiq-työkalulla rautalankamalli (Kuvio 2), josta hahmottuu hyvin sovelluksen perus idea ympäristöjen vertaamiseen. Malli perustui perinteiseen C# ohjelmaan, mikä tehdään Windows Form mallia käyttäen. Malliin tehtiin kaksi esimerkkiä tietojen esittämisestä, jotka voi tehtiin ylä- ja alatasoille taulukossa: kaksi ja kolme riviä. Asennustila väritettäisiin sen sisällön perusteella kuten esimerkissä Installation: OK vihreäksi ja Aborted/Failed punaiseksi.



Kuvio 2: Rautalankamalli

Työpöytäsovelluksen suunnitteluvaiheessa tuli selväksi, että hanke tarvitsee myös tietokannan. Tietokanta mahdollistaa datan helpomman manipuloinnin ja järjestämisen ilman, että tarvitsisi turvautua tekstitiedostojen parsimiseen muussa kuin tietokannan alustamisvaiheessa. Tietokanta myös monimutkaistaisi sovelluksen toteutusta, koska tietokanta rakennettaisiin olemassa olevalle tietokantapalvelimelle ja siihen yhteyden ottaminen työasemista lisäisi haastetta infrastruktuurin sekä tietoturvan kannalta.

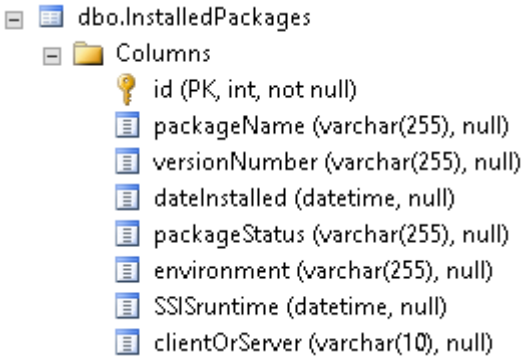
Työpöytäsovellus vaihtui web-sovellukseen suunnitteluvaiheen ja alustavan kehittämisen jälkeen. Kehitysprosessi olisi nopeampi aiemman verkkosivujen kehityskokemuksen vuoksi, sekä se vastaisi moneen haasteeseen palvelin puolella, joita oli noussut työpöytäsovelluksen suunnittelu vaiheessa. Myös helpompi jaettavuus loppukäyttäjille oli suuri tekijä, koska WSS-tukiryhmän jäsenet pääsisivät ympäristödataan käsiksi ilman erillisiä asennuksia. Web-sovellus noudattaisi jo suunniteltua käyttöliittymää.

7.2 Tietokanta

Versionseurantasovellus perustuu enimmäkseen datan esittämiseen, joten tietokanta on sovelluksen tärkeimpiä osia. Tietokannan suunnittelu ja toteutus tehtiin hankkeen toteutuksen alkuvaiheessa, joka mahdollistaa web-sovelluksen kehittämisen sen ympärille. Tämä mahdollistaa oikean datan kanssa työskentelyn ja se nostaa esille todelliset haasteet.

Tietokantana hankkeessa toimii SQL Server 2012, joka pyörii Windows Server 2012 R2 palvelimella. Tietokanta tehtiin valmiille tietokanta-palvelimelle, johon kaikki tarvittavat komponentit, kuten Integration Services, oli jo asennettu. Tietokanta päätettiin toteuttaa yhtenä tauluna, koska dataa ei ole paljon ja sen hakeminen esittämistä varten on paljon yksinkertaisempaa yhdestä taulusta. Relaatiomallinen tietokanta tiivistäisi datan määrää ja mahdollistaisi pakettien historia seuraamisen, mutta tämä ei ollut vaatimuksena toteutuksessa.

Tietokannan taulu (Kuvio 1) luotiin SQL-skriptillä, koska SQL Serverin ohjastettu toiminto ei toiminut odotetulla tavalla, kun haluttiin implementoida id-kentän automaattinen kasvu. Id:n automaattinen kasvu tehdään SQL Serverissä lausekkeella “Id int IDENTITY(1,1) PRIMARY KEY”. Taulun rakenne tehtiin mukailemaan WSS-järjestelmän tuottamaa taulun muotoa. Tämä yksinkertaistaa datan lisäämistä tauluun SQL Serverin Integration Servicesin avulla.



Kuvio 3: InstalledPackages tietokantataulu

Taulu kehittyi vielä myöhemmin, kun siihen lisättiin SSISruntime ja clientOrServer kolumnit Integration Services paketin kehitystyön yhteydessä. Koska tarve huomattiin aikaisessa vaiheessa, oli muutokset helppo tehdä, kun tietokannan datalla ei ollut vielä väliä. Muutokset tehtiin myös SQL-skriptillä, koska alkuperäistä taulunluontiskriptiä oli helppo muuttaa.

7.3 Integration Services -paketin tekeminen

WSS-järjestelmän tuottaman tiedon parsintaan tekstitiedostosta käytettiin Microsoftin SQL Server Integration Services (SSIS) ohjelmaa. Se valittiin tehtävään, koska sen käyttö oli laajaa muissa projekteissa, mikä tarjosi suuremman osaamisen ja informaation työkalusta. Sen avulla pystytään lukemaan sisään tiedot tietokantaan ja lisäämään tarvittavia lisätietoa. SSIS luo .dtsx-muotoisia tiedostoja, joita kutsutaan SSIS-paketeiksi. Ne sisältävät ajettavan logiikan XML muotoisena, joka koostuu kahdesta osasta: Control ja Data flow:sta.

Data Flow-tehtävän suunnittelu aloitettiin ensin, koska tekstitiedostojen sisäänluku on hankkeessa SSIS-paketin oleellisin tehtävä. Tekstitiedostot haetaan Flat File Source-tehtävällä,

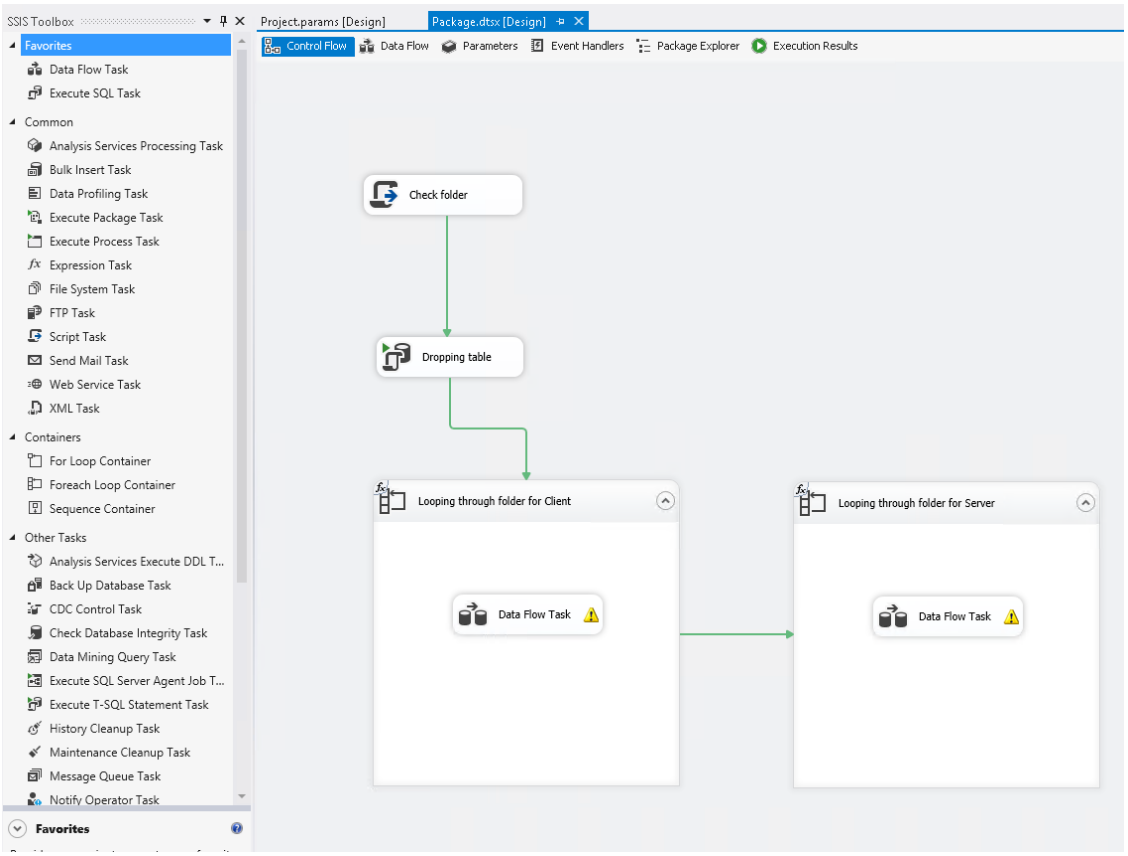
jossa määritellään uusi yhteys tiedostoon Flat File Connection Managerilla. Managerissa tekstitiedostoa käsitellään. Siinä erotellaan sarakkeet ja data määrittämällä erotin, joka katkaisee sarakkeen. Tässä tapauksessa tabulaattori, eli sarkain, oli hyvä merkki erottimeksi, koska sitä ei esiinny tiedon sisällä tai sarakkeiden nimissä.

Seuraavaksi määriteltiin Derived Column Transformation -tehtävä, millä muokataan dataa SQL-funktioilla. Koska WSS-järjestelmän tuottama data sisältää erottelun jälkeen vielä sarkain-merkkejä, täytyy ne trimmata pois TRIM()-funktiolla. Tehtävässä luotiin myös uudet sarakkeet aikaleimalle ja ympäristön arkkitehtuurille, joka ovat muotoa client tai server. Lopulta data viedään tietokantaan SQL Server Destination -tehtävällä, missä määritetään tietokantayhteys. Tietokantayhteys ilmaistaan ConnectionString-attribuuttina, joka on muotoa “Server=PalvelimenNimi;Database=TietokannanNimi;Trusted_Connection=True;”. Tehtävät nimettiin kuvaavasti (Kuvio 4), jotta jatkokehitys olisi helpompaa.

Kuvio 4: Data Flow tehtävä SSIS-paketissa

Paketin pääasiallista kulkua kuvastaa Control Flow, jonka tehtävänä on käydä läpi useita tekstitiedostoja tiedostojärjestelmästä. Data Flow on osa Control Flow:ta ja verrattuna Data Flow käsittelee vian yhtä tiedostoa. Control Flow (Kuvio 5) alkaa C# skripti-tehtävällä, jossa tarkistetaan, onko parametrilla annettu kansio olemassa ja sisältääkö se tiedostoja. Jos ehdot täyttyvät edetään seuraavaan tehtävään, mikä on SQL komento tietokantaan. Komennolla poistetaan kannassa oleva vanha tieto, koska tässä toteutuksessa vanhoja tietoja ei tarvita. Alun tarkistustehtävä on tärkeä, että olematonta tietoa ei yritetä laittaa kantaan, eikä vanhaa tietoa poisteta.

Tiedon poistamisen jälkeen siirrytään kahteen Foreach Loopiin, jotka käyvät läpi parametrilla annetun kansion tekstitiedostot yksitellen. Yksittäisille tiedostoille suoritetaan edellä tehty Data Flow tehtävä, joka suodataa ja lisää datan tietokantaan. Myös Control Flow:ssa tehtäville pyryttiin antamaan kuvaavat nimet.



Kuvio 5: Visual Studio Intergrated Services -työkalu

Toteutuksen edetessä huomattiin, että Visual Studio 2014 tehdyt SSIS-paketit eivät olleet yhteensopivia SQL Server 2012 kanssa vaikka käytössä olivat samat tehtävät kuin vanhimmissa versioissakin. Myöskään vanhemmat versiot Visual Studiosta ja Data Toolsista eivät suostuneet avaamaan uusia SSIS-paketteja. Microsoftin CodePlexistä ladattu ohjelma, mikä muuntaisi uuden SSIS-paketin vanhaan sopivaksi, ei toiminut. Tämän johti logiikan kopioimisen toiselta ruudulta toiselle vanhempaan Data Tools versioon. Tämän olisi voinut välttää tarkemmalla ympäristömäärittelyllä.

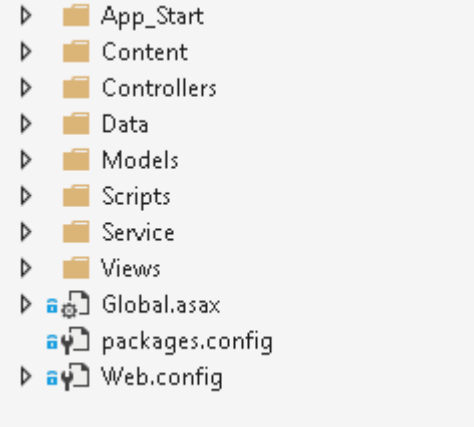
7.4 Web-sovelluksen palvelinpuoli

Web-sovelluksen kehitys alkoi tietokannan valmistuttua ja kun siihen oli saatu konkreettista dataa, jonka kanssa työskennellä. Web-sovelluksen kehittämiseen käytettiin Microsoftin ASP.NET frameworkia. Se toimii luonnollisesti hyvin yhteen Microsoftin muiden tuotteiden

kanssa ja sen avulla sovelluksen asentaminen intranetin web-palvelimelle onnistuu suoraviivaisesti. ASP.NET mahdollistaa .NET Frameworkin vahvempien puolien hyödyntämisen kuten aluksin suunnitellussa työpöytäsovelluksessa pystyisi.

ASP.NET kehitykseen käytettiin Visual Studiota. Käytettäväksi ASP.NET malliksi valittiin ASP.NET MVC, koska ympäristöt ja paketit sisältävät samankaltaista dataa, joten on järkevää käyttää tietojen esittämiseen vain muutamaa sivua, jonka sisältöä voidaan vaihtaa annettavien pyyntöjen mukaan. Tämän MVC mahdollistaa paremmin kuin muut ASP.NET mallit.

MVC-malli on tarkka kansiorakenteesta (Kuvio 13), koska sen perusteella tapahtuu reititys. Visual Studio generoi kehitysprosessin nopeuttamiseksi MVC-mallissa tarvittavan kansiorakenteen ja asentaa myös valmiiksi Bootstrapin ja jQueryn sivuston tyylittelyä varten. Useimmat kansiosta ovat vapaasti poistettavissa ja siirrettävissä, mutta MVC-malli vaatii Controllers ja Views kansiot toimiakseen.



Kuvio 6: Sovelluksen kansiorakenne

Sovelluksessa eriytettiin tietokantayhteys Data kerrokseen ja sovelluksen logiikka Service kerrokseen. Eriytys tapahtui tekemällä tiedostorakenteeseen uudet kansiot ja lisäämällä luokat niiden sisälle. Näin luokille saadaan omat nimiavaruutensa. Tämä myös tekee Controllereista paljon kevyemmät, kun logiikka hoidetaan eri osiossa (Stephen 2009).

7.4.1 Data kerros sovelluksessa

Data kerros (Data layer) pitää sisällään tietokantalogiikan. Tietokantayhteyttä varten tehtiin DatabaseContext-luokka (Kuvio 14). Luokka periytettiin Enity Frameworkin DbContext-luokkasta, jotta saadaan sen tarjoamat tietokanta ominaisuudet käyttöön. Yksi näistä ominaisuuksista on DbSet, jolla tietokannan taulut pystytään yhdistämään sovelluksen modeleihin. Toteutuksessa käytettiin yhtä tietokantataulua InstalledPackages, joka luominen kuvattiin aikaisemmin.

```
namespace WahtiSovellus.Data
{
    public class DatabaseContext : DbContext
    {
        public DatabaseContext() : base("databaseContext")
        {
        }
        public DbSet<InstalledPackages> InstalledPackages { get; set; }

    }
}
```

Kuvio 7: Data layerin DatabaseContext luokka

DatabaseContext() toiminnon avulla saadaan yhteys tietokantaan. Sen avulla haetaan web.config tiedostosta connectionString muuttuja, jossa on tarvittavat tiedot yhteyden muodostamiseksi. ConnectionString on myös ASP.NET:ssä muotoa ”Server=PalvelimenNimi;Database=TietokannanNimi;Trusted_Connection=True;”, joten SSIS-paketissa määriteltyä yhteyttä voidaan hyödyntää.

7.4.2 Service kerros sovelluksessa

Service kerros (Service layer) pitää sisällään sovelluksen businesslogiikan ja mahdollistaa Controllerien pitämisen siisteinä. Projektin näkökulmasta kerros sisältää sovelluksessa käytettävän logiikan datan esittämisen osalta. Tämä sisältää erilaisten taulujen muodostamista tietokannasta haetusta tiedosta. Kansiossa olevalla luokalla on metodeita, joita sovelluksen Controller voi käyttää. Tieto palautetaan Controllerille, joka ohjaa tiedon vieweihin selaimen näytettäväksi.

Service layerin luokka nimettiin WssServiceksi. Luokan alussa määritetään yksityinen olio nimeltä ”_db” Data layerissa olevasta DatabaseContext luokasta. Yksityinen olio ei ole muille luokille nähtävissä, mikä lisää sovelluksen turvallisuutta. Olion avulla pääsemme käyttämään tietokantaa Service layerin WssService luokassa (Kuvio 15). Nyt sovelluksessa tietokantaan pääsee vain tämän kerroksen kautta. Service layerin metodit palauttavat modelin, jotka Controller vie niihin osoitettuihin viewehin esitettäväksi käyttäjälle käyttöliittymässä.

```
public class WssService
{

    //Initialize databaseContext
    private readonly DatabaseContext _db;

    public WssService() : this(new DatabaseContext()) { }

    public WssService(DatabaseContext context)
    {
        _db = context;
    }
```

Kuvio 8: Tietokantaan yhdistäminen Service layerillä

Suurin osa luokan metodeista ovat yksinkertaisia hakuja tietokannasta, kuten viimeisten asennettujen pakettien hakeminen kaikista ympäristöistä (Kuvio 16) tai yhden yksittäisen paketin tietojen hakeminen. Suurimman suunnittelu- ja kehitystyön vaati sovelluksen tärkeimmän ominaisuuden eli kaikkien pakettien hakeminen tietokannasta vertailutauluun.

```
public async Task<IEnumerable<InstalledPackages>> GetLatestAsync()
{
    var latestModel = (from a in _db.InstalledPackages
                       orderby a.dateInstalled descending
                       select a).Take(5).ToArrayAsync();

    return await latestModel;
}
```

Kuvio 9: Viimeisempien tietojen hakemisen metodi

Vertailutaulukko, joka kuvattiin suunnittelu vaiheessa (Taulukko 1), muodostui lopulta monimutkaisella LINQ kyselyllä. Kysely sisältyy omaan metodinsa, joka saa parametrina ”client” tai ”server”, koska vertailutaulu kostuu jommastakummasta. Kyselyssä kaikkien ympäristöjen pakettien tiedot haettiin omiin muuttujiin. Kaikkien ympäristöjen pakettien nimistä valittiin oikeaa kolumnia varten vain eroavat nimet.

Lopulta kaikki punottiin yhteen SQL:sta tutulla LEFT JOIN tyylisellä lauseella, joka jättää myös tyhjät tiedot taulukkoon. LEFT JOIN kuitenkin tehdään LINQ kielessä eri tavalla kuin normaalit JOIN lauseet. Siihen käytetään Where kyselyä, johon määritellään liitettävät kentät kahdesta eri muuttujasta. Tätä voi verrata SQL kielen LEFT JOIN a.id = ON b.id tapaan.

7.4.3 Controllerit

Sovelluksessa Controllereita on kaksi: HomeController ja TableController. HomeController on vastuussa sovelluksen kotinäkymän esittämisestä. Se on yleinen tapa MVC kehityksessä sovelluksen aloitukselle (Freeman 2013, 19). Kun sivustolle saavutaan MVC reitittää käyttäjän HomeControllerin Index Action metodin kautta.

TableController on vastuussa sivuista jotka sisältävät tauluja. Tämä käsittää siis sovelluksen loput sivut kotinäkymän lisäksi. Jokaisen Controllerin aluksi alustetaan ”_wss ”muuttuja, jolla päästään käyttämään WssServicen metodeja. Alustus on hyvin samanlainen kuin tietokanta yhteyden luomisessa Service layerillä.

Koska sovelluksessa käytetään erillistä luokkaa metodien säilyttämiseen, on esimerkiksi TableController todella kevyt ja sen syntaksi on helpommin luettavaa. Sen sisältämät action metodit, kuten ympäristöjen tietojen hakemiseen käytettävä Environment action metodi (Kuvio 17) tarvitsee vain käyttää WssServiceä palauttaakseen tarvittavan modelin viewiä varten. Näin controllerit ovat vastuussa vain sovelluksen kulun logiikasta ja datan haun logiikka on muualla. (Stephen 2009)

```
[Route("{typeOf}/{env}")]
public async Task<ActionResult> Environment(string env, string typeOf)
{
    var environment = await _wss.GetEnvironmentAsync(env, typeOf);

    if (!environment.Any())
    {
        return HttpNotFound();
    }

    ViewBag.Title = typeOf + " / " + env;
    ViewBag.Current = typeOf+env;
    return View(environment);

}
```

Kuvio 10: TableControllerin Environment action metodi

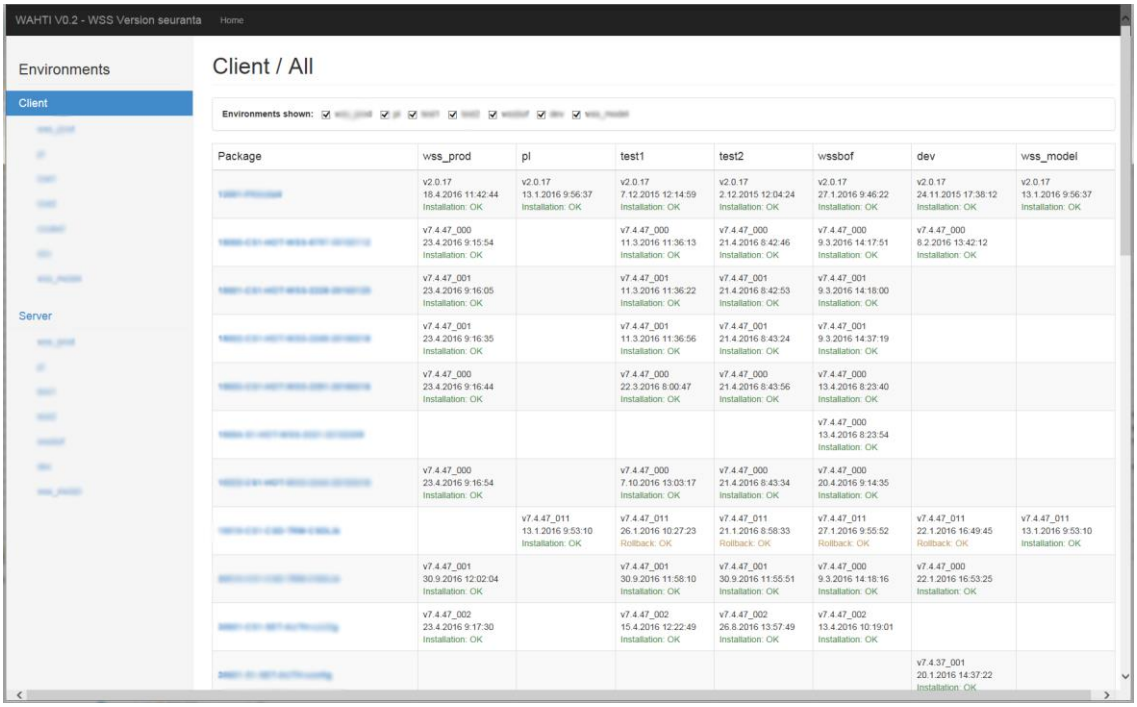
7.5 Web-sovelluksen käyttöliittymä

Sovelluksen käyttöliittymä perustuu vahvasti Bootstrapin tarjoamiin tyyleihin. Bootstrap oli helppo valinta sovelluksen CSS frameworkiksi, koska se tulee valmiina uudessa ASP.NET pro-

jektissa Visual Studion asentamana ja sen kanssa työskentelystä on tutkijalla eniten kokemusta. Valintaan vaikutti myös Bootstrapistä löytyvät suuret määrät verkkomateriaalia ongelmien ratkomiseksi.

Ennen käyttöliittymän kehittämistä selvitettiin mahdollisuudet käyttää valimista pohjaa. Tähän tarkoitukseen löytyi pohja nimeltä Dashboard. Dashboard on ilmainen Bootstrapin tekemä esimerkkipohja, joka löytyy osoitteesta http://getbootstrap.com/examples/dashboard/. Valmiin pohjan käyttäminen nopeutti huomattavasti sovelluksen kehitysprosessia. Pienillä muutoksilla sovelluksen ulkonäöstä saatiin tarpeen mukainen.

Sovelluksen käyttöliittymärakenne on hyvin yksinkertainen. Se koostuu neljästä erilaisesta sivusta: etusivusta, client ja server vertailutauluista, yksittäisestä ympäristöstä, sekä paketti kohtaisista sivuista. Vertailutaulut (Kuvio 6) vastaavat pitkälti alkuperäistä käyttöliittymä suunnitelmaa, joka tehtiin hankkeen alussa. Client ja server vertailutaulut ovatkin sovelluksen tärkeimmät sivut sovelluksen käyttötarkoitusten määritellyn takia. Ne koostuvat kaikista ympäristöistä ja niihin asennetuista paketeista.



Kuvio 11: Client vertailunäkymä

Alkuperäisen suunnitelman mukaan pakettien asennustila tulisi värittää sen sisällön mukaan. Väritys tehdään jQuery funktiolla (Kuvio 7), joka suoritetaan, kun sivun lataus on valmis: $(document).ready(). Jokainen status tieto on ympäröity html-elementillä, joka sisältää luokan ”status”. JQuery funktio etsii sivuston kaikki elementit, jossa on ”status”-luokka

ja lisää span elementtiin Bootstrapin text-success, text-warning tai text-danger luokan, riippuen tekstin sisällöstä. Suunnittelu vaiheessa ei otettu huomioon ”Rollback” tilaa, joka tarkoittaa, että asennus oli vedettävä takaisin. Tämä päätettiin värjätä keltaiseksi, jotta ne olisi helppo erottaa onnistuneista ja epäonnistuneista asennuksista.

```
$(document).ready(function () {

    $(".status:contains('OK')").each(function () {

        $(this).attr("class", "status text-success");
    });

    $(".status:contains('Rollback')").each(function () {

        $(this).attr("class", "status text-warning");
    });

    $(".status:contains('Failed')").each(function () {

        $(this).attr("class", "status text-danger");
    });
```

Kuvio 12: jQuery funktio tilatietojen tyylittelyyn

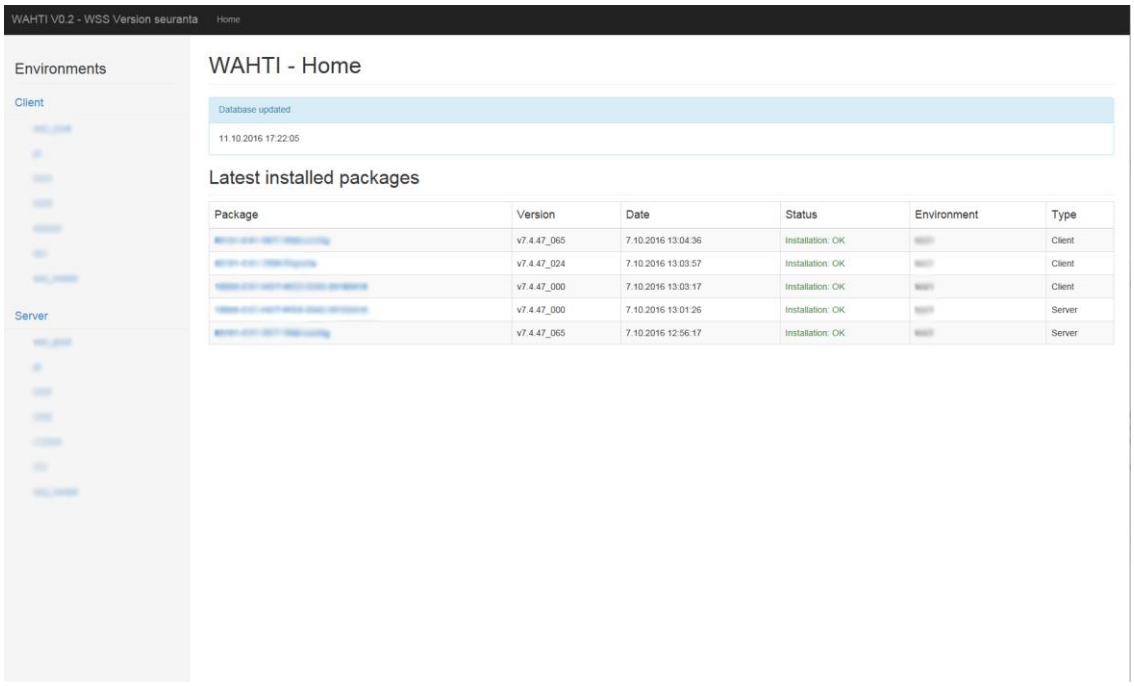
Toiminnallisuutena sivulla on ympäristöjen poistaminen näkyvistä. Ominaisuutta ei alun perin ollut suunniteltu, mutta sen tarpeellisuus huomattiin alustavassa käytössä. Tämän avulla voidaan tarkentaa vielä enemmän ympäristöjen vertailua. Piilottaminen tapahtuu ruksaamalla valintaruutuja taulukon yläpuolelta (Kuva 6). Kun valintaruutu ruksataan, suoritetaan valueChanged() funktio (Kuivio 8), joka tarkistaa, mitkä valintaruudut ovat valittuina ja sen perusteella piilottaa sitä vastaavan ympäristön. Jokaisen ympäristön solu sisältää ympäristön nimeä vastaavan luokan, minkä avulla jQuery hakee, mitkä solut tulee piilottaa (Kuva 8).

```
function valueChanged() {

    if ($('#[redacted]').is(":checked"))
        $(".[redacted]").fadeIn("fast");
    else
        $(".[redacted]").fadeOut("fast");

    if ($('#[redacted]').is(":checked"))
        $(".[redacted]").fadeIn("fast");
    else
        $(".[redacted]").fadeOut("fast");
```

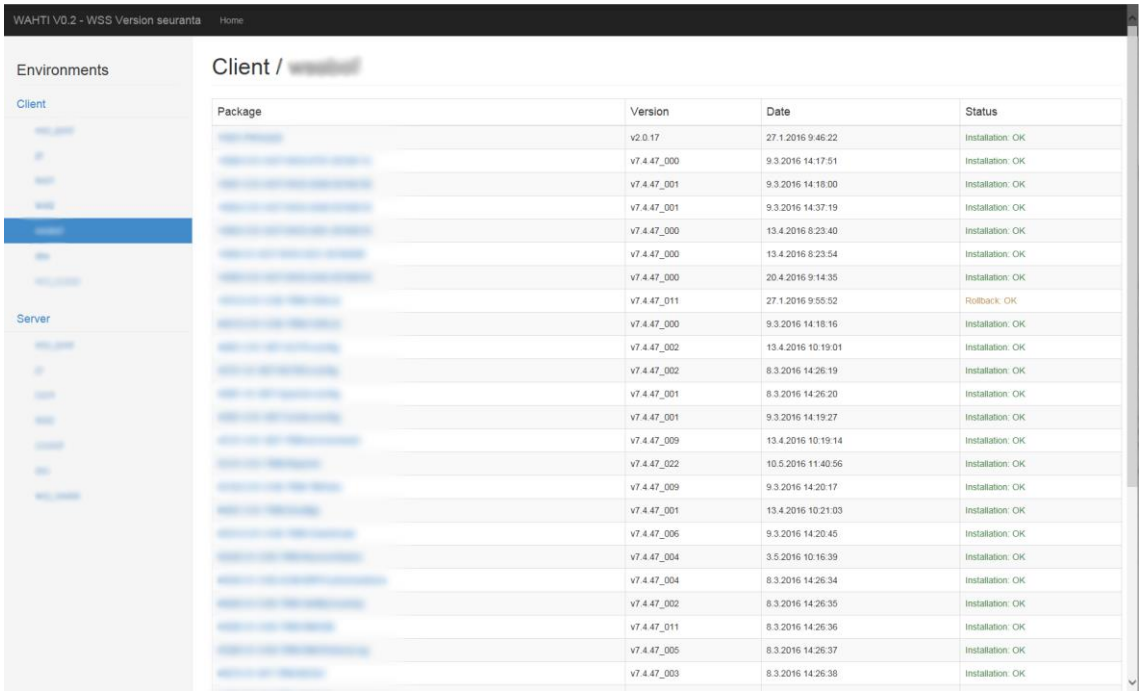
Kuvio 13: Osa jQuery funktiosta taulukon solujen piilottamiseen

Uusimpien asennusten seuraaminen on helppoa etusivulta. Etusivulla (Kuvio 9) näytettään myös tietokannan päivitysaika, mikä tulee SSIS-paketin suoritusajasta. Päivitysajasta on helppo havaita, jos sovelluksen päivityksessä on ollut ongelmia. Yksittäisen ympäristön sivulla

(Kuvio 2) pystyy tarkkailemaan ympäristön asennettuja paketteja tarkemmin. Jokainen paketin nimi on linkki sen paketin yksittäiseen näkymään



Kuvio 14: Sovelluksen etusivu

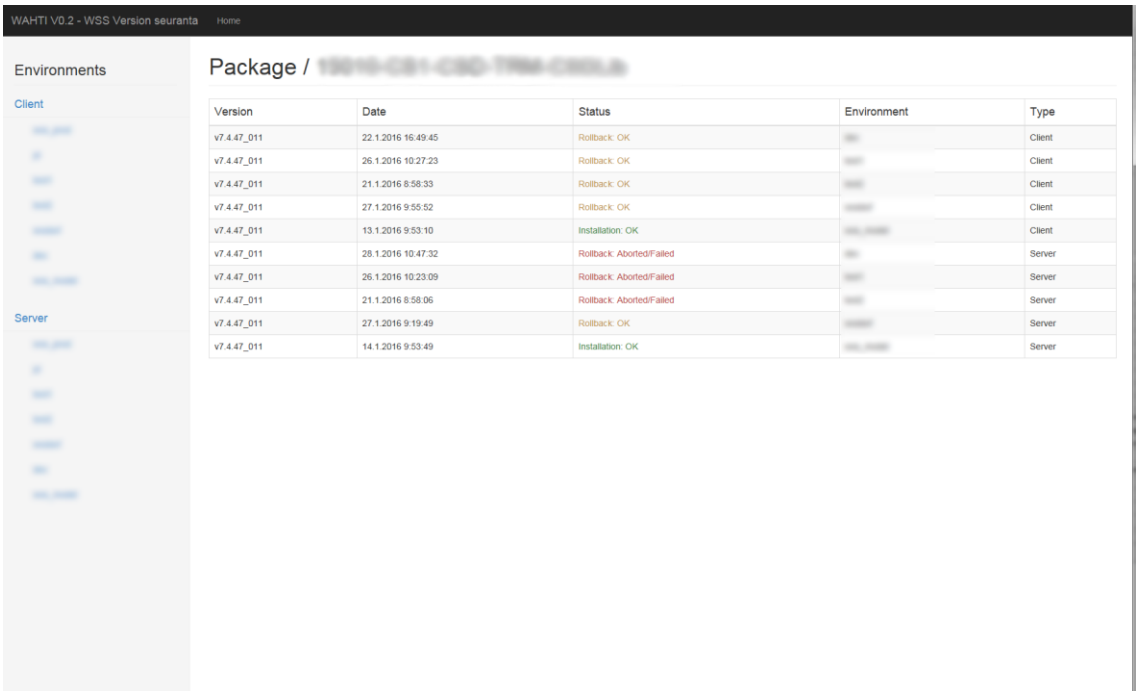


Kuvio 15: Yksittäisen ympäristön sivu

Yksittäisen ympäristön sivulla huomataan yksi palvelinpuolen koodaamisen hyödyistä, kun tietoja pystytään generoimaan pienellä pätkällä koodia ja lisäämään siihen tarvittavat attribuutit HTML sivujen yhdistämiseen (Kuvio 11). Linkki kootaan Razorin ActionLink ominaisuudella modelin jokaisesta elementistä. Siitä otetaan paketin nimi linkin nimeksi ja lähetetään kutsu TableControllerille kasaamaan sivu paketin nimen perusteella (Kuvio 12).

```
@foreach (var item in Model)
{
    <tr>
        <td class="tablePackages">@Html.ActionLink(item.packageName, "Package",
            new { controller = "Table", name = item.packageName })</td>
        <td>@item.versionNumber</td>
        <td>@item.dateInstalled</td>
        <td class="status">@item.packageStatus</td>
</tr>
}
```

Kuvio 16: Palvelinpuolen koodin hyödyntäminen linkeissä

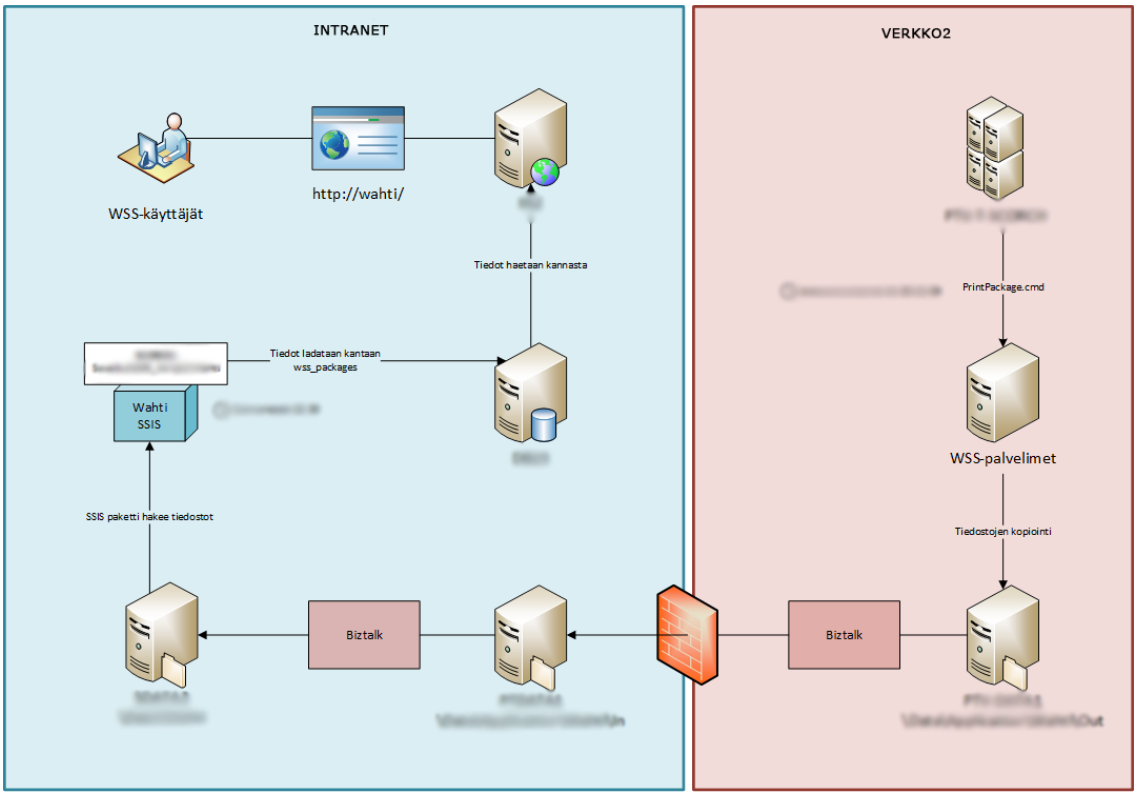


Kuvio 17: Yksittäisen paketin sivu

7.6 Tiedon haun automatisointi

Toteutuksen tärkeimpiä ominaisuuksia on prosessin automatisointi. Tiedon haun prosessissa on monta tekijää ennen kuin tieto saadaan WSS-pankkijärjestelmästä käyttäjän näkyville. Pro-

sessi lähtee liikkeelle pakettien tietojen hakemisesta usealta WSS-palvelimelta. Tiedot haetaan automaattisesti ajettavalla skriptillä, joka käy läpi jokaisen palvelimen yksitellen läpi. Tämä toteutettiin yhteistyössä WSS-ylläpidon kanssa.



Kuvio 18: Arkkitehtuuri kuvaus

Kun tiedot ovat saatu haettua palvelimelta siirretään ne Biztalkilla pankkitoimintaverkosta intranetiin. Intranetissä suoritetaan tiedostoille SSIS-paketti, joka lukee tiedoston datan sisään tietokantaan tietokantapalvelimelle. Tästä päästään tilanteeseen, jossa web-sovellus pystyy lukemaan tiedot tietokannasta.

Automaatio työkaluna käytetään Microsoftin System Center Orchestratoria, joka ajaa prosessin läpi ylläpitotarpeiden vaativin väliajoin. Putken voi myös ajaa läpi manuaalisesti Orchestraattoriin tehdystä ajosta. Arkkitehtuuria toteutettiin Biztalkin ja Orchestraattorin asiantuntioiden kanssa, jotta lopputulos sopisi hyvin Suomen Pankin verkkoon sekä toteutus olisi määrittelyjen mukainen.

7.7 Sovelluksen asentaminen tuotantoon

Tuotantoon vienti vaatii Reguest for change (RFC) eli muutospyynnön. Suomen Pankissa seurataan muutoksia ja jokaisen muutoksen pitää mennä muutospyyntöprosessin läpi. Muutospyyntöä varten listattiin tarvittavat muutokset sovelluksen tuotantoon vientiä varten ja sovellus

pakettina tuotantoon valmiina versiona. Tuotantoon siirrossa konfigurointiin myös web-sovelluksen hostaus.

Web-sovelluksen on tarkoitus toimia sisäverkossa. Koska Suomen Pankilla on käytössään muitakin sisäverkon sovelluksia, on sovelluksen hostaamiseen myös järkevää käyttää sama ratkaisua, eli Micosoftin Internet Information Services, IIS:ää.

IIS Mangerissa tehtiin sovellukselle oma Application pool. Application poolille valittiin sopiva .NET Framework 4.5, jota käytettiin sovelluksen kehityksessä. Sille määritettiin myös service-tunnus, jolla sovellus pääsee käyttämään tietokantaa. Service tunnus on Windows käyttäjä, jolle voidaan antaa vain tarpeelliset oikeudet sovelluksen pyörittämiseen.

Sivustoille on omat asetuksensa IIS Managerissa. Asetuksista määriteltiin web-sovellukselle HTTP-osoite. Tähän tarvitaan olemassa oleva domain name eli verkkotunnus. Pankin sisäverkolla on oma Domain Controller palvelin, joka toimii myös Domain Name Servicenä (DNS). DNS:ssä tehdään uusi alias palvelimelle, jossa web-sovelluksen IIS-verkkopalvelu toimii. Aliakseksi annettiin wahti, joka liitettiin IIS Managerissa web-sovellukseen. Nyt sovellukseen pääsee osoitteella http://wahti/ sisäverkossa.

Sovelluksen julkaisun pystyy tekemään suoraan Visual Studiosta, mutta koska se pitäisi saada kehitysympäristöstä tuotannonympäristöön päädyttiin yksinkertaiseen kansion kopiontiin. Visual Studio kokoama inetpub kansio kopioidaan kokonaisuudessa. Muutoksina tuotantoon siirrossa ei tarvitse tehdä kuin web.configin ConnectionString muuttujaan.

Lopuksi sovellukseen pääsy rajattiin vain WSS-käyttäjille ja ylläpitäjille. Nämä ryhmät tulevat Domain Controller palvelimen Active Directorystä. Oikeuksien lisääminen tapahtuu tiedostojärjestelmästä NTFS-oikeuksia muokkaamalla sivuston kansion ominaisuuksia. IIS Managerista otetiin Anonymous Authentication pois, että luvattomat käyttäjät eivät pääse sivustolle. Samalla aktivoitiin Windows Authentication, joka pyytää Windows kirjautumista, kun sivustolle yritetään.

8 Yhteenveto ja johtopäätökset

Web-sovelluksen kehittäminen vaatii ymmärrystä joka asteelta sovelluksen toimintaa, tietokannan rakentamisesta aina ulkoasun suunniteluun asti. Hyödyllistä on myös ymmärtää tietolähteiden toiminta, josta data järjestelmään alun perin tulee. Tässä hankkeessa käytiin läpi vaiheet web-sovelluksen kehittämisestä aina sen taustan infrastruktuurin hahmottamiseen asti asiantuntijoiden kanssa. Koko ketjun ymmärtämisestä oli paljon hyötyä lopullisen tuotoksen valmiiksi saattamisessa. Se myös antoi paljon kokemusta tulevaisuuden projekteihin.

Alkuvaiheen suunnittelu ja määrittely vaihe osoittautui tärkeäksi suunnannäyttäjäksi sovelluksen kehitysvaiheessa. Tarkemminkin sovelluksen suunnitelman olisi voinut määritellä, mutta lopulta yksinkertainen idea ja rautalankamalli veivät pitkälle. Suurena hyötynä oli, että sai tehdä paljon yhteistyötä WSS-ylläpidon kanssa kasvotusten. Tällöin muutoksista voitiin keskustella sujuvasti ja muutoksia pystyttiin tekemään nopealla aikavälillä. Palaute oli melkein reaaliaikaista.

Hankkeen jälkeen ja projektin valmistuttua nousi esiin eri ideoita mihin sovellusta voisi tulevaisuudessa käyttää ja mihin suuntaan sitä voisi kehittää. Tämä herätti mielenkiintoista keskustelua ja luodusta yksinkertaisesta sovelluksesta saatiin aikaan liikettä muidenkin asioiden kehittämiseksi WSS-ylläpidossa. Jatkokehitystä tehtiin jo hankkeen valmistuttua, jota ei tähän opinnäytetyöhön enää raportoitu.

Sovelluksen jatkoa ajatellen kehitysehdotuksena olisi rakentaa API-rajapintoja, joiden kautta sovellus pystyisi kommunikoimaan suoraan taustajärjestelmiin ilman erillisiä joka päiväisiä tiedosto ajoja. Rajapinnan rakentaminen WSS-järjestelmän eteen mahdollistaisi nykyisen prosessin yksinkertaistamisen ja reaaliaikaisen tiedon päivittämisen sovellukseen taustajärjestelmän rajoitteiden puitteissa. Rajapintaa voisi myös käyttää tulevaisuudessa muihinkin hankkeisiin, eikä pelkästään tämän versionseurannan sovelluksen tarpeeseen.

Kehitetystä web-sovelluksesta tuli hyvä alusta, jonka päälle rakentaa WSS-ylläpidon tarpeiden mukaan uusia ominaisuuksia järjestelmän parhaaksi tukemiseksi. Vaikka kaikki ratkaisut eivät olleet hienostuneita, ne auttoivat sovelluksen valmistua järkevässä aikataulussa. Hankkeen valmistuttua web-sovellus vastasi hyvin sille asetettuihin tavoitteisiin ja WSS-ylläpidon vaatimuksiin.

Lähteet

Kirjalliset lähteet

Freeman, A. 2013. Pro ASP.NET MVC 5. Apress.

Hexter, E., Palermo, J.,Skinner, J., Bogard, J. & Hinze, M. 2012. ASP.NET MVC 4 in Action.

Hirsijärvi, S., Remes, P. & Sajavaara P. 2004. Tutki ja kirjoita. Gummerus : Jyväskylä.

Salonen, K. 2013. Näkökulmia tutkimukselliseen ja toiminnalliseenopinnäytetyöhön : Turun ammattikorkeakoulu.

Shklar, L. & Rosen, R. 2009. Web Application Architecture: Principles, protocols and practices. John Wiley & Sons, Ltd.

Spurlock, J. 2013. Bootstrap.

Sähköiset lähteet

Bootstarp Blog. 2017. Bootstrap 4 Alpha 6.
https://blog.getbootstrap.com/2017/01/06/bootstrap-4-alpha-6/

Bloomberg. Company Overview of Wall Street Systems Delaware, Inc. Viitattu 4.5.2017.
http://www.bloomberg.com/research/stocks/private/snapshot.asp?privcapid=27129735

Bychkov, D. 2013. Desktop vs. Web Applications: A Deeper Look and Comparison.
http://www.seguetech.com/desktop-vs-web-applications/

Collin, M. 2015. Why desktop apps are making a comeback. Viitattu 21.5.2017.
https://medium.com/@collinmathilde/why-desktop-apps-are-making-a-comeback-5b4eb0427647

ION Group. Treasury management : Wallstreet Suite. Viitattu 4.5.2017.
https://iongroup.com/governments/

Kovalick, A. 2014. Understanding Web Apps, SaaS for Media.
http://search.proquest.com.nelli.laurea.fi/docview/1702330516/fulltext/B9BDD51FF1F444BAPQ/1?accountid=12003

Microsoft. 2010. ASP.NET MVC 3: The Razor View Engine. Viitattu 14.5.2017.
https://docs.microsoft.com/en-us/aspnet/mvc/mvc3#the-razor-view-engine

Microsoft. Getting Started with the .NET Framework. Viitattu 4.5.2017.
https://msdn.microsoft.com/fi-fi/library/hh425099(v=vs.110).aspx

Microsoft. SQL Server Integration Services. Viitattu 4.5.2017.
https://msdn.microsoft.com/en-us/library/ms141026(v=sql.110).aspx

Microsoft. SSIS Catalog. Viitattu 4.5.2017.
https://msdn.microsoft.com/en-us/library/hh479588(v=sql.110).aspx

Mozilla Developer Network. JavaScript Guide: Introduction. Viitattu 4.5.2017.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

Stephen, W. 2009. Validating with a Service Layer (C#). Viitattu 14.5.2017.
https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/models-data/validating-with-a-service-layer-cs

Stevens, A. 2014. Bootstrap: You're Doing It Wrong. Viitattu 21.5.2017.
http://alanstevens.io/blog/bootstrap-youre-doing-it-wrong/

Suomen Pankki. Suomen Pankin vuosikertomus 2015. Viitattu 4.5.2017.
https://helda.helsinki.fi/bof/bitstream/handle/123456789/14033/VK_FI_2015.pdf?sequence=1&isAllowed=y

Suomen Pankki. Hallinto-osasto. Viitattu 4.5.2017.
https://www.suomenpankki.fi/fi/suomen-pankki/organisaatio/hallinto-osasto/

Techterms. 2013. Framework. Viitattu 4.5.2017.
https://techterms.com/definition/framework

Virtuaali ammattikorkeakoulu. Monimuotoinen / Toiminnallinen opinnäytetyö. Viitattu 4.5.2017.
http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/030906/1113558655385/1154602577913/1154670359399/1154756862024.html

Virtuaali ammattikorkeakoulu. Miksi opinnäytetyö? Tarkoitus ja tavoitteet. Viitattu 4.5.2017.
http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/030906/1113558655385/1113561758365/1154602189329/1154602747549.html

W3C. 2014. HTML5: A vocabulary and associated APIs for HTML and XHTML. Viitattu 4.5.2017.
https://www.w3.org/TR/html5/introduction.html#background

W3C. HTML & CSS. Viitattu 4.5.2017.
https://www.w3.org/standards/webdesign/htmlcss

W3Techs. 2017. Usage of server-side programming languages for websites. Viitattu 20.5.2017.
https://w3techs.com/technologies/overview/programming_language/all

W3Techs. 2017. Technologies Overview. Viitattu 20.5.2017.
https://w3techs.com/technologies

Kuviot